Manual del usuario: PaRTiKle sobre sistemas LPC2000

S. Peiró
Rev: Miguel Masmano
Grupo de Informática Industrial y Sistemas de Tiempo Real
Universidad Politécnica de Valencia, Spain
{speiro,mmasmano}@ai2.upv.es

30 de enero de 2012

Resumen

En este texto se describe la puesta a punto desde cero del entorno de programación para desarrollar aplicaciones que se ejecuten sobre los sistemas LPC2000 con el SOTR PaRTiKle [**?**]. Dentro del entorno de programación se contempla tanto la parte hardware como la parte software:

Hardware El sistema LPC2000 y los dispositivos necesarios para tener acceso desde una terminal serie (*host*) hasta el LPC2000 (*target*).

Software El sistema operativo de tiempo real para sistemas empotrados PaRTiKle [**?**], las herramientas de compilación (*toolchain*) y el cargador por puerto serie utilizado para cargar código en el LPC2000.

Keywords: PaRTiKle, Real Time Operating Systems, LPC200, ARM7, System Programming

Índice

1. Sistemas LPC2000

Este apartado presenta una visión general de los sistemas utilizados, proporcionando referencias a documentos más específicos que quedan fuera del alcance de este manual. El manual se centra en particular en el modelo LPC2136 que utiliza los procesadores de la arquitectura ARM (*Advanced RISC Machines*) de 32 bits, en particular la familia ARM7TDMI-S, donde la S final hace referencia a que el procesador esta modelado en VHDL.

1.1. Características LPC2000

A continuación se muestran las características generales del modelo LPC2136:

Modelo	LPC2136
Procesador	ARM7TDMI-S 60 Mhz
Memoria RAM	32 Kbytes (expandible módulo externo)
Memoria ROM	256 Kbytes EEPROM (electrically-erasable programmable ROM)
Puertos serie	UART0: serial (38400 bauds) + UART1: modem
Relojes	RTC 32.768 KHz
Timers	t0, t1: 15Mhz (CCLK= 60 Mhz / PDIV = 4)
Otros	gpio, spi, pwm, i2c, can, ...

Cuadro 1: Resumen de especificaciones LPC2136

En la figura **??** se puede ver la placa utilizada en el robot de tipo humanoide: *µbiro v2* (20 º de libertad para el movimiento del robot) este robot incorpora el sistema LPC2136 sobre el que se ha portado el sistema operativo `PaRTiKle`.

La placa ha sido diseñada en el Grupo de Informática Industrial en el Instituto de Automática e Informática industrial (http://www.ai2.upv.es). Se puede encontrar más detalles, así como videos del robot *µbiro v2* en funcionamiento en la siguiente dirección http://www.gii.upv.es/personal/mialgil/ubiro2.htm

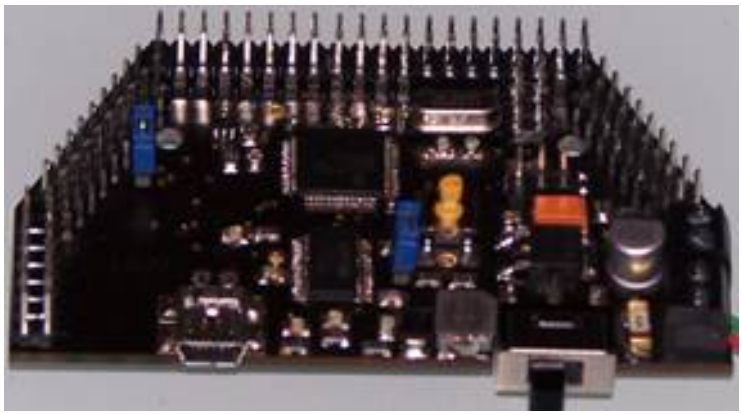


Figura 1: Placa utilizada por *µbiro v2*

Para una descripción más detallada de las características del LPC2136 se puede ver su manual del usuario [**?**], así como el manual del procesador ARM7TDMI [**?**].

Por otra parte para cuestiones generales sobre la arquitectura ARM ver el *Architecture Reference Manual* [**?**].

A continuación a modo de introducción se presentan una serie de secciones que tratan los aspectos básicos de la programación de los sistemas LPC2000. En estas secciones se da respuesta a dos preguntas básicas: el procedimiento para arrancar al procesador y el procedimiento de carga de código para su posterior ejecución.

1.1.1. Inicialización

Al poner en funcionamiento el procesador, este comienza a ejecutar código en el vector de interrupción de reset `vector[reset]`. Abajo se pueden ver las distintas causas de interrupción que puede recibir el procesador, junto con los vectores de interrupción a cargar en cada caso y sus direcciones de memoria:

# entradas	# direcciones	# propósito
vector[reset]	0x00000000	Reset
vector[undf]	0x00000004	Undefined Instruction
vector[swi]	0x00000008	Software interrupt
vector[pabt]	0x00000010	Program abort
vector[dabt]	0x00000014	Data abort
vector[cksum]	0x00000018	Reserved: checksum
vector[irq]	0x0000001c	Interrupt ReQuest
vector[fiq]	0x00000020	Fast Interrupt reQuest

Figura 2: Vectores de interrupción ARM

Tras un reset el procesador asigna al contador de pograma (PC) la dirección del vector de interrupción de reset `vector[reset]`. Como resultado de esto se comienza a ejecutar la instrucción que se encuentre en `PC = 0x00000000`. El `vector[reset]` es el encargado de inicializar el LPC2000 en el siguiente orden:

- Código de arranque, programación PLL, ...
- Inicialización de los modos de interrupción, reservar pilas de programa (stacks), ...
- Interrupciones, configurar las interrupciones para su uso.
- Dispositivos: en particular el puerto serie para poder depurar el código que estamos ejecutando.

Inicialización de los modos de interrupción Las restantes entradas del vector de interrupción también deben de inicializarse, ya que se encargan de la gestión de interrupciones (IRQ, FIQ) y excepciones: (Undefined, Data Abort, Program Abort, etc.) que es interesante tratar para presentar un volcado del estado del procesador: registros y pila, para tener información suficiente para comprender las causas de la excepción.

Reserva de pilas de programa El estándar del procedimiento de llamadas a funciones de C para la arquitectura ARM ATPCS [**?**], requiere de un puntero a una pila de memoria para proporcionar los bloques de activación (almacenamiento de variables locales, temporales, ...) de las funciones en C.

Para ello es necesario durante la inicialización del procesador proporcionar una zona de memoria suficientemente grande para la pila del modo funcionamiento del procesador SVC.

En caso de no ser así, y si el tamaño reservado para dicha pila fuera inferior al utilizado, el sistema comenzaría a utilizar zonas de memoria no reservada, con lo que dejaría de funcionar correctamente. En dicho caso se pueden editar los fuentes para ampliar el tamaño de la pila SVC.

1.1.2. Interrupciones

Para gestionar los dispositivos de los LPC2000 se utiliza el modelo de peticiones de interrupción, de forma que el procesador se encuentra en su hilo de ejecución, y cuando alguno de los dispositivos requiere la atención del procesador, realiza una petición de interrupción que el procesador puede atender. Para ello el procesador ARM unicamente proporciona los modos de funcionamiento IRQ y FIQ, el resto es tarea del sistema específico.

Así pues los sistemas ARM incorporan un controlador de interrupciones, que se encarga de notificar al procesador de las interrupciones y proporcionar una interfaz para su acceso/control desde el procesador. En el caso del LPC2000, este controlador se implementa mediante el VIC (*Virtual Interrupt Controller*).

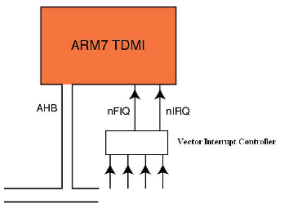


Figura 3: Controlador VIC LPC2000

El VIC presenta un conjunto de registros (mapeados en la memoria del ARM7) para acceder/controlar las interrupciones de los distintos dispositivos disponibles en el LPC2000, por medio de escrituras/lecturas en memoria. Cuando se recibe una interrupción el procesador carga en el PC la entrada correspondiente del vector de interrupción (`vector[irq]`). Esta a su vez será una instrucción de salto o modificación del PC, en el caso de las IRQ, FIQ se trata de un salto a un manejador de interrupciones global encargado de determinar la interrupción del dispositivo específico que se ha producido y ejecutar su manejador asociado.

1.1.3. Uart serie

Los LPC2000 incorporan dos uarts en el chip, las dos son idénticas en cuanto a su modo de uso, con la excepción de que la UART1 proporciona un soporte adicional para el uso de un MODEM. Ambos periféricos cumplen la especificación “estándar industrial 550” [**?**]. Ambas contiene un generador de baud rate y fifos de 16 bytes para agilizar la recepción/transmisión de carácteres.

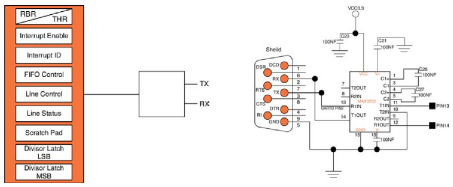


Figura 4: UARTs serie presentes en el LPC2000

1.1.4. Gestión de tiempo

Los LPC2000 incluyen para la gestión del tiempo un reloj (RTC) y dos temporizadores (T0 y T1).

El RTC (*Real Time Clock*) que funciona a una frecuencia de 32Khz como reloj del sistema. La resolución de este reloj es de $(1/32Khz) = 3{,}05 \cdot 10^{-5}$ sec.

Los temporizadores T0 y T1 pueden ser utilizados tanto como temporizadores como contadores. Estos funcionan a la frecuencia del bus de periféricos PCLK (*Peripheral bus Clock*) que se calcula de la siguiente forma:

$$CCLK = 60 \cdot 10^6 = 60Mhz$$
$$PDIV = 4$$
$$PCLK = CCLK/PDIV = 15 \cdot 10^6 = 15Mhz$$

Que proporciona una resolución de $6{,}66 \cdot 10^{-8}$ sec.

1.1.5. JTAG/ETM

Un aspecto importante de estos dispositivos es que proporcionan interfaces para facilitar las tareas de depuración/testing, las cuales suelen ser necesarias en las etapas iniciales de programación. En el caso de LPC2000 se puede utilizar un debugger como GDB/Insight/OpenOCD para acceder al dispositivo a traves de la interfaz JTAG (*Joint Test Action Group*): http://www.google.es/search?q=LPC2000+with+JTAG+gdb+insight

En caso de no disponer de debugger, se utiliza la depuración por medio de `printf()` por el puerto serie.

2. Entorno de programación

En este apartado se comenta la puesta a punto desde cero de un entorno de programación para desarrollar aplicaciones sobre el sistema operativo embarcado PaRTiKle.

En este apartado se contemplan las tareas de:

- Configuración del compilador,
- Configuración del cargador flash,
- Configuración de `PaRTiKle`,
- Y por último la compilación y ejecución de aplicaciones sobre `PaRTiKle`.

2.1. Configuración compilador

Para programar el LPC2000 es necesario tener un conjunto de herramientas (compilador, linker, ensamblador, ...) que permitan generar código para el juego de instrucciones Arm/Thumb del procesador. Para ello se utiliza el compilador cruzado (host: x86, target: arm) del proyecto GCC, que permite generar código para el procesador arm7tdmi-s.

El compilador utilizado para compilar `PaRTiKle` para el LPC2000 se ha obtenido de http://www.mikrocontroller.net [**?**]. Los pasos para preparar el entorno de compilación utilizado por `PaRTiKle` asumen que el directorio raiz del compilador se encuentre en `/usr/local/arm/lpc`. Podemos descargar los fuentes del compilador con los siguientes comandos:

```
$ # fuente: http://www.mikrocontroller.net
$ wget http://www.mikrocontroller.net/download/\
      arm-toolchain-linux-2.tar.bz2
$ tar xjvf arm-toolchain-linux-2.tar.bz2 -C /usr/local/
$ mv /usr/local/arm /usr/local/arm/lpc
```

Una vez realizados estos pasos, los ejecutables del compilador son referenciados por `PaRTiKle` (desde `partikle/mkfiles/mkfile-lpc`), antes de continuar podemos comprobar que el compilador se ha instalado correctamente, con el siguiente comando, que debería mostrar la ruta completa a los ejecutables: gcc, ld, as

```
$ ls /usr/local/arm/lpc/bin/arm-elf-{gcc,ld,as}
/usr/local/arm/lpc/bin/arm-elf-as
/usr/local/arm/lpc/bin/arm-elf-gcc
/usr/local/arm/lpc/bin/arm-elf-ld
```

2.2. Configuración cargador flash

Para cargar programas, el LPC2000 proporciona dos protocolos:

- ISP: *In System Programming*
- IAP: *In Application Programming*

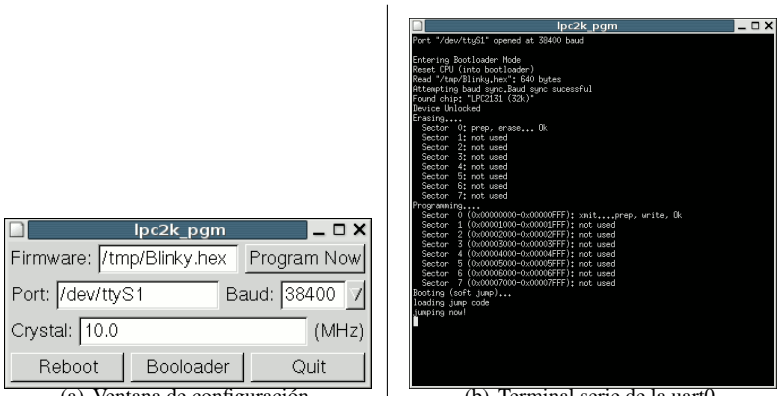
Durante la fase de arranque, el LPC2000 inicializa el puerto serie y comprueba si en el otro extremo está preparado para iniciar el protocolo ISP (ver el apartado the "Flash memory system and programming" [**?**]). Mediante el ISP, los cargadores envian

el código al LPC2000 que se encarga de almacenarlo en la EEPROM y a continuación ejecutarlo.

Existen varios cargadores para Unix, los dos más comunes y utilizados son: lpc2isp (http://www.cmucam.org/browser/trunk/tools/lpc21isp) y lpc2k_pgm. En nuestro caso vamos a utilizar este último. `lpc2k_pgm` [**?**] presenta como ventajas una interfaz gráfica y el incorporar la emulación de una terminal serie. Esto evita tener que utilizar un programa de emulación de terminal externo.

Es importante ajustar correctamente los parámetros de funcionamiento del cargador, en particular los campos más relevantes son:

- **Crystal**: La frecuencia del reloj utilizado por el procesador, al consultar el manual del LPC2136 encontramos que es de 10 MHz.
- **Baud**: Tras varias pruebas con el cargador, se ha comprobado que la mayor frecuencia de funcionamiento del puerto serie es de 38400 baudios (se pueden encontrar más detalles en el manual sobre el motivo de esta frecuencia de funcionamiento).
- **Port**: El puerto serie del PC (*host*) al que se ha conectado el LPC2000.



(a) Ventana de configuración (b) Terminal serie de la uart0

Figura 5: Ventana de configuración y terminal serie de la uart0 del LPC2000

Tras ajustar correctamente los parámetros del cargador, al ejecutar la orden de programar, nos aparecerá una ventana de terminal por la que se vuelca la salida del puerto serie UART0 del LPC2000.

Como se puede observa en la imagen **??**, la extensión del fichero cargado es ".hex". Esto se debe a que el contenido del fichero imagen cargado en la ROM debe estar en formato ihex (*Intel Hexadecimal*) [**?**].

Todos nuestro programas deberán ser traducidos del formato ELF (formato por defecto producido por el linker en el entorno Linux) a ihex. Para ello disponemos de la herramiento `objcopy`.

2.2.1. Conexión serie con FTDI

Por la parte hardware es necesario disponer de un cable USB a mini USB. Para poder acceder al UART serie que incorpora el sistema LPC2000, la conexión serie se proporciona a traves de USB por medio de un driver FTDI [**?**]. Por ello es necesario

disponer de un driver USB con soporte para FTDI en el host (PC) que se está utilizando como plataforma de desarrollo. Este driver permitirá usar la conexión serie. En el caso de GNU/Linux se ha utilizado el módulo usbserial (con soporte FTDI) que trás cargarse proporciona el dispositivo /dev/ttyUSB0, a traves del que se puede comunicar a traves del puerto serie.

2.3. Configuración de PaRTiKle

Una vez instalado correctamente el compilador cruzado, podemos pasar a obtener y configurar PaRTiKle [**?**].

La versión más actualizada de PaRTiKle está disponible en el repositorio del grupo de investigación (https://www.gii.upv.es/svn/rtos/trunk/partikle). Los fuentes de PaRTiKle pueden residir en cualquier directorio. En nuestro caso, supondremos /usr/src/partikle.

```
$ cd /usr/src/
$ svn co https://www.gii.upv.es/svn/rtos/trunk/partikle
$ cd /usr/src/partikle
$ make menuconfig
```

Los aspectos a tener en cuenta durante la configuración son la arquitectura y la memoria dinámica disponible en el sistema:

```
Architecture
    >> LPC (arm7) Stand Alone system
Core Options
    >> Kernel Dynamic Memory: 18432 bytes (18 Kbytes)
    >> User Dynamic Memory: 10240  bytes (10 Kbytes)
    >> Thread stack size: 512  bytes (0.5 Kbytes)
```

Por lo tanto, se tiene que ajustar el valor THREAD_STACKSZ, durante la configuración de PaRTiKle para adaptarlos a los requisitos de la aplicación, en este caso la memoria RAM del LPC2000 es de 32 Kbytes por lo que 512 bytes de pila per thread puede ser un buen valor por defecto (ver figura **??**).

Tras realizar las anteriores selecciones podemos guardar los cambios (en el fichero .config) y salir.

El resultado tras salir será:

```
>> Building the configuration utility:...

>> End of PaRTiKle kernel configuration <<
>> Execute 'make' to build the kernel <<
```

Tras esto, basta con ejecutar la orden make para compilar PaRTiKle.

```
$ make
>> Detected PaRTiKle path: /usr/src/partikle
>> Building PaRTiKle utils: done
>> Building PaRTiKle kernel [lpc]: done
>> Building PaRTiKle user libraries: done

>> Include these in your profile environment:
   PRTK=/usr/src/partikle export PRTK
   PATH=$PATH:$PRTK/user/bin export PATH
```

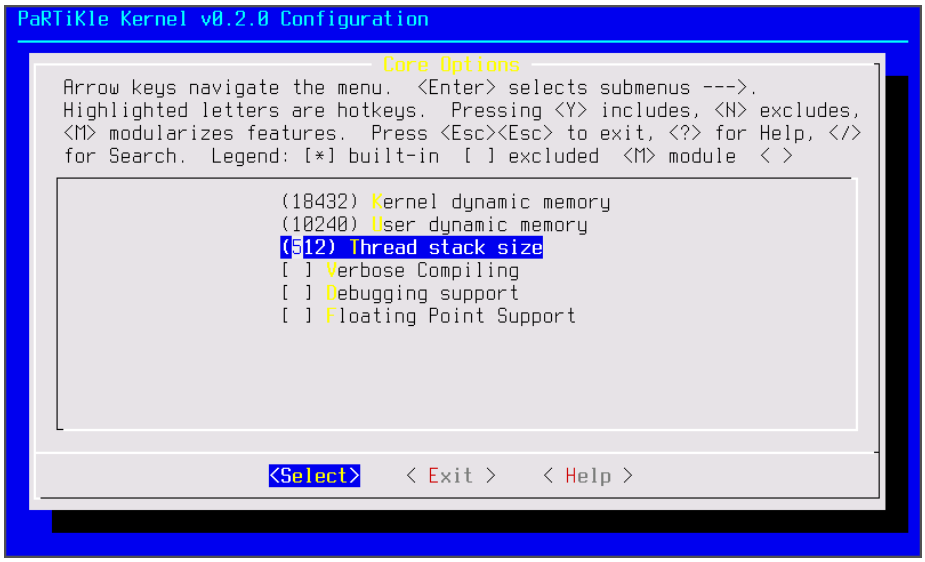

PaRTiKle Kernel v0.2.0 Configuration
Core Options
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module < >
(18432) Kernel dynamic memory
(10240) User dynamic memory
(512) Thread stack size
[ ] Verbose Compiling
[ ] Debugging support
[ ] Floating Point Support
<Select> < Exit > < Help >

Figura 6: Configuración de PaRTiKle bajo el LPC2136

2.3.1. Introducción a `PaRTiKle`

La interfaz básica que ofrece es la de POSIX threads PSE52 (subconjunto de servicios POSIX para sistemas mínimos). Una característica muy interesante de `PaRTiKle` es su modularidad, esto es, `PaRTiKle` implementa y ofrece la interfaz POSIX-threads basándose en una interfaz con el hardware genérica. Con lo que es extremadamente sencillo "portar" `PaRTiKle` a distintas arquitecturas.

Buena muestra de esto es el porting de PaRTiKle a la arquitectura ARM7 para sistemas LPC que se describe en este documento. Los aspectos más destacados que facilitan las tareas de porting a nuevas arquitecturas son:

1. Conjunto reducido de requisitos hardware:
   - initcode: código de inicialización.
   - interrupts: manejo de interrupciones.
   - timers: programación de timers/clocks.
   - print: rutinas para enviar carácteres a un dispositivo de salida (vga, serie, ...)
2. Proceso de configuración y compilación sencillo.
3. Estructura del código fuente:

   En el momento de programar una nueva funcionalidad para el sistema, se debe determinar si esta es común a todo el sistema (por ejemplo, planificador, gestión timers, señales, sistema de ficheros, ...) en dicho caso el nuevo códigos será código portable y residirá en el directorio `core/kernel/port`.

   Si por lo contrario se trata de código específico para una arquitectura este ocupa un directorio propio: `core/kernel/$arch` por cada arquitectura *$arch*.

2.3.2. Características de PaRTiKle para LPC

Todas las cuestiones asociadas al LPC2000 has sido ya mencionadas anteriormente.

Es importante remarcar que en la implementación se ha intentado minimizar la cantidad de código ensamblador escrito, para ello se ha utilizado únicamente en los detalles de inicialización donde era imprescindible utilizar ensamblador. Como resultado utilizando la herramienta SLOCcount (SLOC: *Source Lines Of Code*) se puede comprobar que:

```
% cd /usr/src/partikle/core/kernel/lpc/
% sloccount .
SLOC    Directory       SLOC-by-Language (Sorted)
1511    lpc             ansic=1410,asm=101

Totals grouped by language (dominant language first):
ansic:          1410 (93.32%)
asm:             101 (6.68%)
```

Básicamente, el código ensamblador escrito se encarga de preparar un contexto para poder llamar a la primera función C, para llevar a cabo el resto de la inicialización por medio de código escrito en C, por medio de lecturas/escrituras sobre registros.

2.3.3. Programación de aplicaciones PaRTiKle

Una vez terminado el anterior paso, ya podemos utilizar `PaRTiKle` para desarrollar aplicaciones con el LPC2000. A modo de ejemplo podemos compilar y ejecutar los ejemplos que se distribuyen con PaRTiKle (se encuentran en `user/examples/c_examples`):.

```
$ cd /usr/src/partikle
$ cd user/examples/c_examples
$ make
```

Tras esto se compilarán los programas ".c" contenidos en el directorio actual, el resultado será un fichero ".prtk" por cada ".c", los ficheros ".prtk" contienen el ejecutable del programa en formato ELF, para poder ser cargados en el LPC2000 utilizando el lpc2k_pgm han de ser convertidos a Intel Hex, lo que se realiza mediante la orden `make`:

```
$ make hello_world.hex
```

para luego utilizar el cargador flash, para transmitirlo al LPC2000 y ejecutarlo, ver la siguiente captura de pantalla (figura **??**) del proceso de carga y ejecución:

```
$ lpc2k_pgm
```

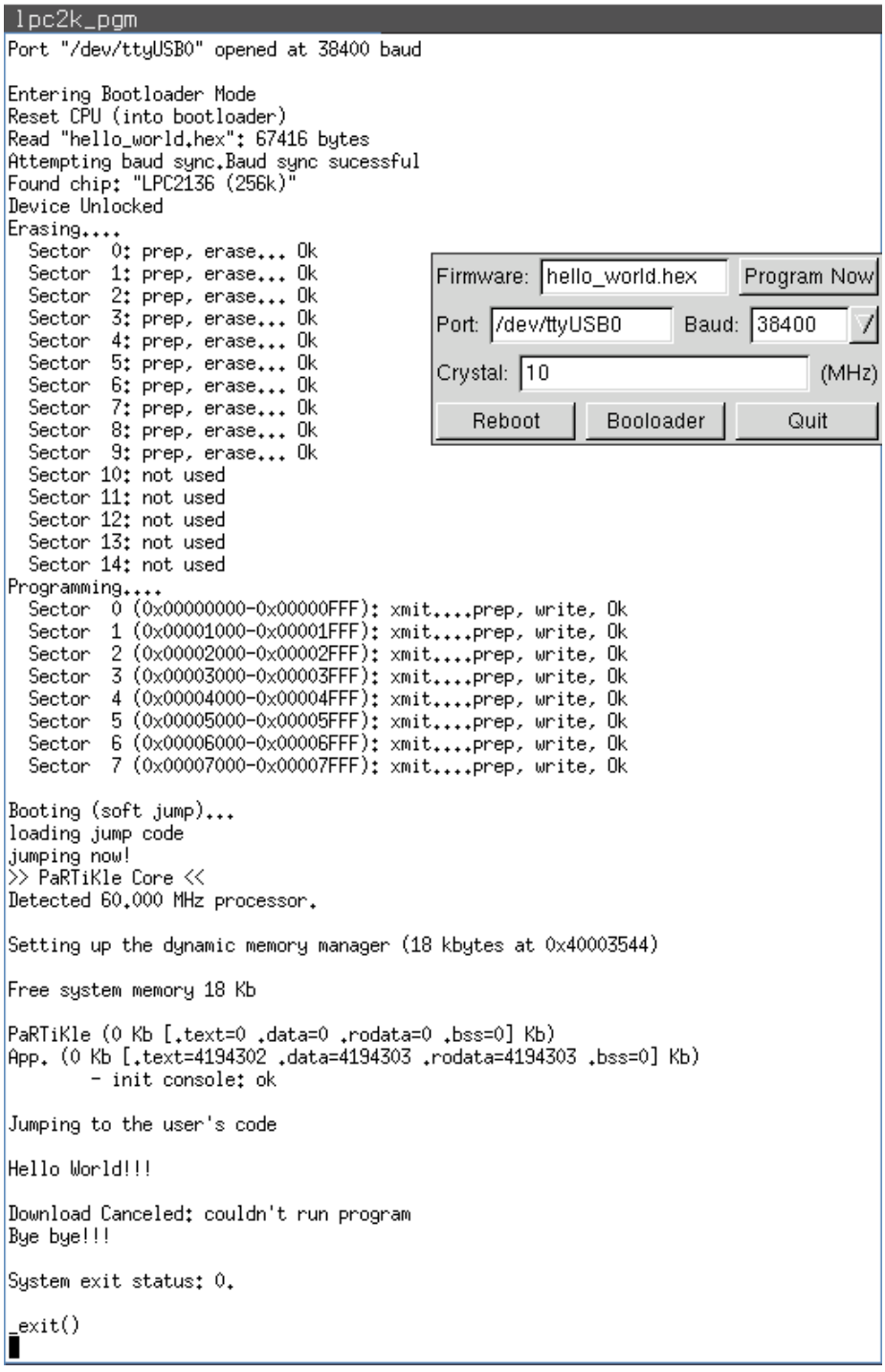

lpc2k_pgm
Port "/dev/ttyUSB0" opened at 38400 baud
Entering Bootloader Mode
Reset CPU (into bootloader)
Read "hello_world.hex": 67416 bytes
Attempting baud sync.Baud sync sucessful
Found chip: "LPC2136 (256k)"
Device Unlocked
Erasing....
Sector 0: prep, erase... Ok
Sector 1: prep, erase... Ok
Sector 2: prep, erase... Ok
Sector 3: prep, erase... Ok
Sector 4: prep, erase... Ok
Sector 5: prep, erase... Ok
Sector 6: prep, erase... Ok
Sector 7: prep, erase... Ok
Sector 8: prep, erase... Ok
Sector 9: prep, erase... Ok
Sector 10: not used
Sector 11: not used
Sector 12: not used
Sector 13: not used
Sector 14: not used
Programming....
Sector 0 (0x00000000-0x00000FFF): xmit....prep, write, Ok
Sector 1 (0x00001000-0x00001FFF): xmit....prep, write, Ok
Sector 2 (0x00002000-0x00002FFF): xmit....prep, write, Ok
Sector 3 (0x00003000-0x00003FFF): xmit....prep, write, Ok
Sector 4 (0x00004000-0x00004FFF): xmit....prep, write, Ok
Sector 5 (0x00005000-0x00005FFF): xmit....prep, write, Ok
Sector 6 (0x00006000-0x00006FFF): xmit....prep, write, Ok
Sector 7 (0x00007000-0x00007FFF): xmit....prep, write, Ok
Booting (soft jump)...
loading jump code
jumping now!
>> PaRTiKle Core <<
Detected 60.000 MHz processor.
Setting up the dynamic memory manager (18 kbytes at 0x40003544)
Free system memory 18 Kb
PaRTiKle (0 Kb [.text=0 .data=0 .rodata=0 .bss=0] Kb)
App. (0 Kb [.text=4194302 .data=4194303 .rodata=4194303 .bss=0] Kb)
- init console: ok
Jumping to the user's code
Hello World!!!
Download Canceled: couldn't run program
Bye bye!!!
System exit status: 0.
_exit()
Firmware: hello_world.hex
Program Now
Port: /dev/ttyUSB0
Baud: 38400
Crystal: 10
(MHz)
Reboot
Booloader
Quit

Figura 7: Arranque de PaRTiKle bajo el LPC2136

Referencias

[1] S. Peiro, M. Masmano, I. Ripoll, and A. Crespo, *PaRTiKle OS, a replacement of the RTLinux core*. Real-Time Systems Group, Universidad Politécnica de Valencia. http://www.e-rtl.org/partikle/node/5, http://www.e-rtl.org/partikle.

[2] M. Masmano, I. Ripoll, A. Crespo, *PaRTiKle OS User Manual*. Real-Time Systems Group, Universidad Politécnica de Valencia. http://www.e-rtl.org/partikle/node/5.

[3] M. Masmano; I. Ripoll, A. Crespo, *An overview of the XtratuM nanokernel*, Workshop on Operating Systems Platforms for Embedded Real-Time applications, (2005), http://www.xtratum.org/biblio.

[4] NXP, *LPC2131/2/4/6/8 User manual*. NXP, founded by Philips, http://www.standardics.nxp.com/support/documents/microcontrollers/?search=LPC2&type=user.

[5] Trevor Martin, *The Insider's guide to the ARM7-based microcontrollers*. Hitex development tools, http://hitex.co.uk/arm.

[6] Advanced RISC Machines, *ARM7TDMI-S (rev r4p3) Technical Reference manual*. ARM Limited. http://www.arm.com/documentation/ARMProcessor_Cores.

[7] David Seal, *The ARM Architecture Reference Manual*. 2nd Edition, Addison-Wesley Longman Publishing Co. http://www.arm.com/documentation/books.html.

[8] NXP, *LPC2000 Application Notes, Boot sequence, Interrupts, Spurious Interrupts*. NXP, founded by Philips. http://www.standardics.nxp.com/support/documents/microcontrollers/?search=LPC2.

[9] Paul Stoffregen, *LPC2K_PGM Linux Bootloader Utility (Philips LPC 2000 ARM7 Chips)*. http://www.pjrc.com/arm/lpc2k_pgm.

[10] *The Mikrocontroller project*, http://www.mikrocontroller.net/articles/LPC2000.

[12] Martin Thomas, *Philips LPC213x/214x examples ported for the GNU-Toolchain*, http://www.siwawi.arubi.uni-kl.de/avr_projects/arm_projects/lpc2k_bundle_port/.

[13] Intel Corporation, *Intel Hexadecimal Object File Format Specification*, Intel 1988, http://www.pjrc.com/tech/8051/pm2_docs/intel-hex.html.

[14] Craig Peacock, *USB with the simplicity of RS-232* http://www.beyondlogic.org/usb/ftdi.htm.

[15] Craig Peacock, *Interfacing the Serial / RS-232 Port* http://www.beyondlogic.org/serial/serial.htm

[16] Development systems Division, Compiler Tools Group *Procedure Call Standard for the ARM Architecture*. ARM Limited, 2003. http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042a/IHI0042A_aapcs.pdf

[17] Leslie Lamport. *LaTeX: A Document Preparation System. (User's Guide and Reference Manual)*. Addison-Wesley, Reading, Massachusetts, second edition, 1986.